UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/591,172	06/09/2000	David Wallman	SUN1P2/P4566	3517

22434 7590 11/10/2003

BEYER WEAVER & THOMAS LLP
P.O. BOX 778
BERKELEY, CA 94704-0778

EXAMINER	
SHAH, NILESH R	

ART UNIT	PAPER NUMBER
2127	

DATE MAILED: 11/10/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/591,172	WALLMAN, DAVID	
	Examiner	Art Unit	
	Nilesh R Shah	2127	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 June 2000* .

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-21* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-21* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *09 June 2000* is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)☐ The proposed drawing correction filed on _____ is: a)☐ approved b)☐ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____ .

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) ☐ The translation of the foreign language provisional application has been received.

15)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) *5* .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in a patent granted on an application for patent by another filed in the United States before the invention thereof by the applicant for patent, or on an international application by another who has fulfilled the requirements of paragraphs (1), (2), and (4) of section 371(c) of this title before the invention thereof by the applicant for patent.

The changes made to 35 U.S.C. 102(e) by the American Inventors Protection Act of 1999 (AIPA) and the Intellectual Property and High Technology Technical Amendments Act of 2002 do not apply when the reference is a U.S. patent resulting directly or indirectly from an international application filed before November 29, 2000. Therefore, the prior art date of the reference is determined under 35 U.S.C. 102(e) prior to the amendment by the AIPA (pre-AIPA 35 U.S.C. 102(e)).

2. Claims1-21 are rejected under 35 U.S.C. 102(e) as being anticipated by Nilsen et al (6,081,665) (hereinafter Nilsen)

3. As per claim 1 Nilsen teaches a computing system, the computing system including: a processor; a memory; and a virtual machine which is in communication with the processor (col. 5 lines 30 –65, col. 29) ('The invention is a real-time virtual machine method (RTVMM) for use in implementing portable real-time systems. The RTVMM provides

efficient support for execution of portable byte-code representations of computer programs, including support for accurate defragmenting real-time garbage collection. Efficiency is measured both in terms of memory utilization, CPU time, and programmer productivity. Programmer productivity is enhanced through reduction of the human effort required to make the RTVMM available in multiple execution environments.')

the virtual machine being arranged to enable one or more jobs to run thereon, wherein the virtual machine is arranged to create a heap in the memory for each job that runs on the virtual machine. (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be

granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

4. As per claim 2, Nilsen teaches a computing system wherein the virtual machine is scaleable (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.')

5. As per claim 3, Nilsen teaches a computing system wherein the virtual machine includes a jobs manager, a class manager, and a heap manager. (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine

implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.')

6. As per claim 4, Nilsen teaches a computing system wherein the heap manager manages substantially all heaps in the memory that are created by the virtual machine. (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') ('Occasionally, application programmers desire to access the elements of an array of a particular type as if certain slots contained elements of a different type. Suppose, for example, that the programmers want to treat the 2nd and 3rd entries of an integer array as a 64-bit integer. This can be achieved using the GetHeapInArrayNonPtr() macro, as demonstrated below:')

7. As per claim 5, Nilsen teaches a computing system wherein the heap manager is arranged to allow an object allocated on a first heap created by the virtual machine to be visible to a second heap created by the virtual machine. (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the

implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') ('Occasionally, application programmers desire to access the elements of an array of a particular type as if certain slots contained elements of a different type. Suppose, for example, that the programmers want to treat the 2nd and 3rd entries of an integer array as a 64-bit integer. This can be achieved using the GetHeapInArrayNonPtr() macro, as demonstrated below:')

8. As per claim 6, Nilsen teaches a computing system wherein the heap manager uses an object router to exchange data between the first heap and the second heap. (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') ('Occasionally, application programmers desire to access the elements of an array of a particular type as if certain slots contained elements of a different type. Suppose, for example, that the programmers want to treat the 2nd and 3rd entries of an integer array as a 64-bit integer. This can be achieved using the GetHeapInArrayNonPtr() macro, as demonstrated below:')

9. As per claim 7, Nilsen teaches a computing system wherein the class manager is arranged to enable a class associated with the virtual machine to be shared by the one or more jobs that are arranged to run on the virtual machine (col. 20 line 15 – col. 21 line 44) ('It is

necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

10. As per claim 8, Nilsen teaches a computing system wherein the virtual machine is further arranged to enable information to be exchanged between the one or more jobs that are arranged to be run on the virtual machine (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is

increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.--list queue is maintained in priority order.')

11. As per claim 9, Nilsen teaches a computing system the virtual machine is created using the Java programming language (col. 12 lines 22- 55) ('PERC (and Java) is an object-oriented programming language. Programs are comprised of object type declarations, known in PERC as classes. Each class definition describes the variables that are associated with each instance (object) of the corresponding class and also defines all of the operations that can be applied to instantiated objects of this type. Operations are known as methods.')

12. As per claim 10 Nilsen teaches a computing system the virtual machine is associated with a system heap, and the virtual machine is further arranged to create a system garbage collector, the system garbage collector being arranged to perform garbage collection on the system heap. (Fig 2, Fig. 32, col. 45 line 39 – col. 44 line 43) ('This section describes the special techniques that are used to implement allocation of objects within the garbage collected heap. Every newly allocated object can be assumed to contain all zeros.') (' Each heap-allocated object must be identified so that the garbage collector can determine which of its fields contain pointers. The standard technique for identifying pointers within heap objects is to provide a signature for each object. The signature pointer occupies a particular word of each object's header (See FIG. 2).')

13. As per claim 11, Nilsen teaches a computing system wherein the virtual machine is further arranged to create an incremental garbage collector for each heap created in the memory, the incremental garbage collector for each heap being arranged to perform garbage collection on its associated heap. (Fig 2, Fig. 32, col. 45 line 39 – col. 44 line 43) ('This section describes the special techniques that are used to implement allocation of objects within the garbage collected heap. Every newly allocated object can be assumed to contain all zeros.') ('Each heap-allocated object must be identified so that the garbage collector can determine which of its fields contain pointers. The standard technique for identifying pointers within heap objects is to provide a signature for each object. The signature pointer occupies a particular word of each object's header (See FIG. 2).')

14. As per claim 12, Nilsen teaches a computing system wherein the one or more jobs are arranged to execute substantially concurrently (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations.

See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

15. As per claim 13, Nilsen teaches a virtual machine arranged to operate in cooperation with a computing system, the virtual machine including: a first mechanism for creating a first

job and a second job, the first job and the second job being arranged to run with respect to the virtual machine; a second mechanism, the second mechanism being arranged to provide the at least one job with at least one class that is arranged to be shared between the first job and the second job; and a third mechanism, the third mechanism being arranged to exchange information between the first job and the second job. (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it

doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

16. As per claim 14, Nilsen teaches a virtual machine wherein the first mechanism is further arranged to create a first heap associated with the first job and a second heap associated with the second job, and the third mechanism is further arranged to increase a size of the first heap and to decrease a size of the second heap (Fig 2, Fig. 32, col. 45 line 39 – col. 44 line 43) ('This section describes the special techniques that are used to implement allocation of objects within the garbage collected heap. Every newly allocated object can be assumed to contain all zeros.') (' Each heap-allocated object must be identified so that the garbage collector can determine which of its fields contain pointers. The standard technique for identifying pointers within heap objects is to provide a signature for each object. The signature pointer occupies a particular word of each object's header (See FIG. 2).')

17. As per claim 15, Nilsen teaches a virtual machine further including a first garbage collector and a second garbage collector, wherein the first garbage collector is arranged to perform a garbage collection on the first heap and the second garbage collector is arranged to perform a garbage collection on the second heap(Fig 2, Fig. 32, col. 45 line 39 – col. 44 line 43) ('This section describes the special techniques that are used to implement allocation of objects within the garbage collected heap. Every newly allocated object can be assumed to contain all zeros.') (' Each heap-allocated object must be identified so that the garbage collector can determine which of its fields contain pointers. The standard technique for identifying pointers within heap objects is to provide a signature for each object. The signature pointer occupies a particular word of each object's header (See FIG. 2).')

18. As per claim 16, Nilsen teaches a virtual machine wherein the size of the first heap and the size of the second heap may be dynamically altered (col. 42 line 35 – col. 43 line 20) ('Native libraries are implemented according to a protocol that allows references to dynamic objects to be automatically updated whenever the dynamic object is relocated by the garbage collector. However, if these native libraries call system routines, which do not follow the native-library protocols, then the system routines are likely to become confused when the corresponding objects are moved. To avoid this problem, programmers who need to pass heap pointers to system libraries must make a stable copy of the heap object and pass a pointer to the stable copy. The stable copy should be

allocated on the C stack, as a local variable. If necessary, upon return from the system library, the contents of the stable copy should be copied back into the heap. Note that on uniprocessor systems a non-portable performance optimization to this strategy is possible when invoking system libraries that are known not to block if thread preemption is under PERC's control. In particular, we can pass the system library a pointer to the dynamic object and be assured that the dynamic object will not be relocated (since the garbage collector will not be allowed to run) during execution of the system library routine.')

19. As per claim 17, Nilsen teaches a virtual machine wherein the second mechanism is further arranged to share the at least one class between the first job and the second job (col. 14 line 15 –col.16 line 44) ('When the method to be invoked is declared as static within the corresponding object (meaning that the method operates on class information rather than manipulating variables associated with a particular instance of the corresponding class), the Java compiler treats this as an invokeStatic method. Execution of static methods is identical to execution of special methods except that there is no implicit pointer to "this" passed as an argument to the called method. See FIG. 47, FIG. 45, and FIG. 46.') ('With findMethod(), the desired method is described by a pointer to the known Class object and a String pointer to the method's name and signature. With getMethodPtr(), the desired method is described by String representations of the class name and of the method's name and signature. Prototypes for both functions are provided below:')

20. As per claim 18, Nilsen teaches a virtual machine wherein at least one of the first job and the second job includes data which is persisted (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority

inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

21. As per claim 19 Nilsen teaches a computer-implemented method for executing a first application substantially concurrently with a second application, the computer-implemented method comprising:

creating a first job on a virtual machine, the first job being associated with the first application, creating a second job on the virtual machine, the second job being associated with the second application (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these

macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.');

creating a first heap, the first heap being associated with the first job; and creating a second heap, the second heap being associated with the second job (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables

between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35 lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

22. As per claim 20, Nilsen teaches a computer-implemented method wherein the first job and the second job share at least one class (col. 14 line 15 –col.16 line 44) ('When the

method to be invoked is declared as static within the corresponding object (meaning that the method operates on class information rather than manipulating variables associated with a particular instance of the corresponding class), the Java compiler treats this as an invokeStatic method. Execution of static methods is identical to execution of special methods except that there is no implicit pointer to "this" passed as an argument to the called method. See FIG. 47, FIG. 45, and FIG. 46.') ('With findMethod(), the desired method is described by a pointer to the known Class object and a String pointer to the method's name and signature. With getMethodPtr(), the desired method is described by String representations of the class name and of the method's name and signature. Prototypes for both functions are provided below:')

23. As per claim 21, Nilsen teaches a computer-implemented method wherein the first application and the second application both executed substantially simultaneously (col. 37 line 15 – col. 38 line 11) (In general, the PERC virtual machine is intended to support many more priority levels than might be supported by an underlying operating system. Further, the design of the real-time application programmer interface (API) is such that task dispatching cannot be fully relegated to traditional fixed priority dispatchers. Thus, the PERC virtual machine implements its own task dispatcher which communicates with an underlying thread model.') (col. 20 line 15 – col. 21 line 44) ('It is necessary to provide parameterized access to heap memory so as to facilitate the implementation of read and write barriers. The following macros serve to copy variables between different kinds of representations. See FIG. 41 for implementations of these macros.') (col. 35

lines 5-53) ('Obtaining and releasing monitor locks. When application code desires to enter a monitor, it executes the enterMonitor instruction. This instruction first consults the object's lock field. If this field is NULL, it allocates a HashLock object, initializes its count field to 1, sets its u.owner field to represent the current thread, and grants access to the newly locked object. If the lock field is non-NULL, enterMonitor examines the contents of the HashLock object to determine whether access to the lock can be granted. If the count field equals 0, or if the u.owner field refers to the currently executing thread, the count field is incremented, the u.owner field is made to point to the current thread if it doesn't already, and access is granted to the newly locked object. Otherwise, this lock is owned by another thread. The current thread is placed onto the waiting.sub.-- list queue and its execution is blocked until the object's lock can be granted to this thread. Priority inheritance describes the notion that if a high-priority thread is forced to block waiting for a low-priority thread to release its lock on a particular object, the low-priority thread should temporarily inherit the priority of the higher priority blocked task. This is because, under this circumstance, the urgency of the locking task is increased by the fact that a high-priority task needs this task to get out of its way. The PERC virtual machine implements priority inheritance. Furthermore, the waiting.sub.-- list queue is maintained in priority order.')

24. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Nilesh R Shah whose telephone number is 703-305-8105. The examiner can normally be reached on Monday-Friday 8am-4pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William Grant can be reached on 703-3058-1108. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.

MAJID A. BANANKHAH
PRIMARY EXAMINER

NS

October 22, 2003